Record Linkage and Entity Resolution for Job Application Data: A Composite Embedding and Semantic Similarity Approach

Harry A Barnish
LucidTrack
harry@lucidtrack.dev

Bernard Lee
LucidTrack
bernard@lucidtrack.dev

December 29, 2025

Abstract

Hundreds of users independently discovering and saving the same job opportunities creates significant data quality challenges: duplicate application entries with varying metadata and inconsistent company name representations. This paper presents a comprehensive framework for record linkage [1] and entity resolution [2] in job application data, addressing these challenges through two complementary mechanisms. We describe a composite embedding-based approach for application grouping that combines semantic representations of company name, position title, and URL using weighted sums, enabling accurate duplicate detection across multiple attributes simultaneously. For company entity resolution, we employ both fuzzy string matching and semantic embedding techniques [3] to normalise company representations while preserving original data. The framework incorporates an interactive administrative workflow that balances automated detection with human oversight, enabling the transformation of fragmented, duplicate-ridden application data into clean, structured datasets suitable for jobs board aggregation. Empirical validation through real-world deployment demonstrates significant improvements in data quality metrics and successful generation of consolidated job postings from user-generated data.

Keywords: Entity resolution, record linkage, semantic similarity, vector embeddings, cosine similarity, composite embeddings, fuzzy string matching

1 Introduction

1.1 The LucidTrack Context

LucidTrack was developed to address a fundamental problem faced by university students and recent graduates: the overwhelming complexity of managing internship and graduate scheme applications. Students typically submit between forty and one hundred and twenty applications for any given role, with each application involving multiple rounds including assessments, interviews, and assessment centres.

The platform provides students with a centralised system for tracking applications, managing deadlines, storing job descriptions, and monitoring their progress through various application stages. Users can create applications through multiple channels: a Chrome browser extension that captures job postings directly from company websites and job boards, manual entry through the web interface, and cloning functionality that allows users to share applications with peers. While this multi-channel approach provides flexibility and convenience, it introduces significant data quality challenges that become apparent as the user base grows and the volume of application data increases.

1.2 Problem Statement

The data quality challenges in LucidTrack manifest in two primary forms: duplicate application entries and inconsistent company representations. These issues arise from the collective behaviour of hundreds of students discovering and saving the same job opportunities through different channels and entry methods.

The core challenge emerges when multiple users discover the same job posting independently. As students browse company websites, job boards, and other sources, they naturally encounter the same opportunities. Each student who discovers a position creates their own application entry, resulting in hundreds of separate records for what is fundamentally the same job opportunity. These entries vary in their data quality and completeness: one student might capture the position title exactly as it appears on the company website, while another might enter a slightly abbreviated version. One student might include the full company legal name, while another uses the common brand name. Location information might be captured differently depending on where the posting was discovered, and application URLs may vary if the same position is listed on multiple platforms.

This proliferation of duplicate entries creates significant challenges for aggregating user data into a coherent jobs board. When hundreds of students have each created their own application entry for the same position, the platform must identify that these entries represent a single opportunity and consolidate them into a unified job posting. Without intelligent grouping, the jobs board would display the same position hundreds of times, each with slightly different metadata, making it impossible for students to discover opportunities effectively. The variation in how different users capture the same information means that simple exact matching cannot identify these duplicates; the system must recognise that "Software Engineer" and "Software Engineer - Full Stack" might represent the same role, or that "Google LLC" and "Google" refer to the same company.

To illustrate this challenge, consider a popular software engineering internship at a major technology company. Over the course of a recruitment season, 150 students might independently discover and save this position. One student might capture it as "Software Engineer Intern - Summer 2024" from the company's careers page, another as "SWE Intern" from a job board, and a third as "Software Engineering Internship" from a university careers service. Each entry would have slightly different company names ("Google", "Google LLC", "Alphabet Inc."), different URLs (company website, job board listing, referral link), and varying location information. Without intelligent grouping, these 150 entries would appear as 150 separate opportunities on the jobs board, overwhelming students and making it impossible to determine which entries represent the same position.

The second major challenge involves inconsistent company representation. The same organisation may appear under various names throughout the system due to legal entity variations, brand name differences, user entry inconsistencies, and international naming conventions. This inconsistent representation creates fragmented company profiles that impede the jobs board's search and filtering capabilities, as users cannot reliably find all opportunities for a company when it appears under multiple names.

1.3 Motivation and Objectives

The data cleansing framework was developed to enable the aggregation of user-generated application data to create a comprehensive jobs board that benefits the entire student community. As students create applications through various channels, they collectively generate a rich dataset of job opportunities, company information, and application details. However, this aggregated data is only valuable if it can be properly normalised and consolidated.

The primary objective of the framework is to transform fragmented, duplicate-ridden application data into a clean, structured dataset that can power a jobs board feature. This jobs board

would allow students to discover opportunities that other users have found, search by company or sector, and benefit from the collective knowledge of the platform's user base. To achieve this, the system must identify when multiple users have discovered the same job opportunity, normalise company representations so that searches and filters work reliably, and consolidate duplicate entries so that the jobs board presents each opportunity once rather than multiple times.

The framework addresses these challenges through two complementary mechanisms. The first mechanism, application grouping, automatically identifies and groups duplicate or highly similar job applications across all users, enabling the creation of consolidated job postings that represent each unique opportunity. The second mechanism, company entity assignment, normalises company representations through a canonical entity system. Together, these mechanisms transform the raw application data into a structured, searchable jobs board dataset.

The system is implemented as an in-house administrative tool used by LucidTrack's development and data quality teams, rather than being directly accessible to end users. This design choice recognises that data cleansing requires expertise in matching algorithms, entity resolution, and data quality assessment. By centralising these operations in an administrative tool, the system can be used to periodically review and clean the database, ensuring that the jobs board and other aggregated features have access to high-quality, normalised data.

1.4 Scope and Contributions

The primary contributions of this work include a composite embedding-based matching algorithm that considers multiple application attributes simultaneously, enabling more accurate identification of duplicate applications than approaches that consider only single attributes. We also present a hierarchical company entity system that maintains canonical representations while preserving original data for audit purposes.

The system incorporates an interactive workflow that combines automated detection with human oversight, recognising that while algorithms can identify potential duplicates and company variations, administrators must ultimately decide how to resolve these issues based on their assessment of data quality and the requirements of the jobs board feature. This workflow is implemented through a dedicated data cleansing interface that presents results in an organised, reviewable format and provides clear actions for consolidation and normalisation.

Empirical validation of the approach through real-world implementation within LucidTrack demonstrates significant improvements in data quality metrics and the ability to generate a comprehensive jobs board from user-generated data. The framework has been deployed as a production system and is actively used by the development team to maintain data quality and prepare aggregated datasets for platform features.

2 Background and Related Work

2.1 Duplicate Detection Techniques

Traditional duplicate detection methods each have strengths and limitations that inform our approach. Exact matching, which performs simple string comparison, is effective only for identical entries and fails to identify duplicates that differ in formatting, capitalisation, or minor variations. Fuzzy matching uses edit distance algorithms such as Levenshtein distance [4] and Jaro-Winkler similarity [5], [6] to find similar strings even when they are not identical, making it effective for handling typographical errors and formatting differences. Semantic matching employs machine learning embeddings [3] to capture semantic similarity between text strings, generating vector representations that capture meaning and context rather than just character-level differences.

Our approach combines fuzzy and semantic matching, allowing administrators to select the most appropriate method for their specific use case. For company name matching, where legal variations and abbreviations are common, semantic matching often provides better results. For application grouping, where we need to match across multiple attributes simultaneously, we employ composite embeddings that combine semantic representations of company name, position title, and application URL.

2.2 Entity Resolution

Entity resolution is the process of identifying records that refer to the same real-world entity, even when those records contain different representations or variations of the entity's identifying information [2]. In the context of LucidTrack, entity resolution involves recognising that different company name variations represent the same organisation, and that different application entries represent the same job opportunity.

The canonical entity model provides a single source of truth for each distinct company or job posting while maintaining links to all variations and representations, enabling normalised operations such as search, analytics, and reporting that benefit from consistent entity representation.

3 System Architecture

3.1 Overview

The data cleansing system consists of three primary components: the application grouping module that identifies and groups similar applications, the company entity assignment module that normalises company representations, and the user interface layer that provides interactive tools for review and action. These components are implemented as a separate in-house data cleansing application that operates on the LucidTrack database. The tool is used exclusively by administrators and data quality specialists within the LucidTrack development team, not by end users of the platform. The system is designed to be non-destructive, meaning that it identifies issues and proposes resolutions without automatically modifying user data. All changes require explicit administrator confirmation.

3.2 Data Model

The system operates on core entities including applications (individual job application records created by users), companies (employer records with names, custom flags, sector classifications, and entity assignments), company entities (canonical representations with standardised names, websites, and sectors), and job postings (consolidated job opportunities created from grouped applications). The distinction between custom and system-generated companies is important for the entity assignment process, as system-generated companies are typically preferred as canonical representations when resolving duplicates.

3.3 Workflow

The system follows a semi-automated workflow that balances algorithmic efficiency with human judgment. Figure 1 illustrates the four-phase workflow: detection, presentation, action, and persistence. The detection phase employs algorithms to identify potential duplicates and company variations, generating similarity scores and grouping candidates based on configurable thresholds. The presentation phase organises results in a hierarchical, reviewable format. The action phase enables administrators to select groups and apply appropriate actions such as creating

consolidated job postings or assigning companies to entities. The persistence phase commits approved changes to the database within transactional boundaries, ensuring atomic updates and maintaining full audit trails.

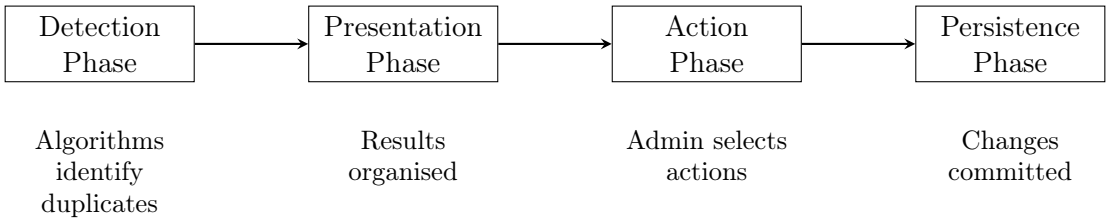

Figure 1: The four-phase workflow of the data cleansing system, showing the progression from automated detection through administrator review to final persistence.

4 Application Grouping

4.1 Purpose and Rationale

Application grouping addresses the fundamental problem of duplicate application entries that arise when hundreds of users independently discover the same job opportunities. By identifying related applications across all users, the system enables the creation of consolidated job postings that represent each unique opportunity accurately. These consolidated postings aggregate metadata from all related application entries, providing the most complete picture possible for the jobs board. When applications are grouped and consolidated into job postings, the original application entries remain linked to the consolidated posting, maintaining a complete audit trail while enabling aggregated features that benefit from consolidated, normalised data.

4.2 Matching Algorithm

The application grouping system employs a composite embedding approach that considers multiple attributes simultaneously. For each application, a composite embedding is generated by combining semantic representations of the company name, position title, and optionally the application URL using a weighted sum. The composite embedding $\mathbf{E}_{\text{composite}}$ is computed as:

$$\mathbf{E}_{\text{composite}} = \frac{w_c \cdot \mathbf{E}_c + w_t \cdot \mathbf{E}_t + w_u \cdot \mathbf{E}_u}{\|w_c \cdot \mathbf{E}_c + w_t \cdot \mathbf{E}_t + w_u \cdot \mathbf{E}_u\|} \tag{1}$$

where $\mathbf{E}_c$, $\mathbf{E}_t$, and $\mathbf{E}_u$ represent the embeddings for company name, position title, and URL respectively, and w_c, w_t, and w_u are their corresponding weights with default values of $w_c = 0.5$, $w_t = 0.3$, and $w_u = 0.2$. The result is normalised to unit length for cosine similarity computation. Figure 2 illustrates how these three embeddings are combined to form the composite representation.

Similarity between applications is computed using cosine similarity [7] on the composite embeddings, which measures the angle between embedding vectors in high-dimensional space:

$$\text{similarity}(A, B) = \cos(\theta) = \frac{\mathbf{E}_A \cdot \mathbf{E}_B}{\|\mathbf{E}_A\|\|\mathbf{E}_B\|} = \frac{\sum_{i=1}^{n} E_{A,i} E_{B,i}}{\sqrt{\sum_{i=1}^{n} E_{A,i}^2}\sqrt{\sum_{i=1}^{n} E_{B,i}^2}} \tag{2}$$

where $\mathbf{E}_A$ and $\mathbf{E}_B$ are the composite embeddings for applications A and B respectively, and θ is the angle between the vectors. The similarity score ranges from 0 to 1, where 1 indicates identical applications and 0 indicates completely dissimilar applications.

The grouping process begins with embedding generation for all applications, followed by construction of a similarity matrix containing pairwise similarity scores computed using Equation

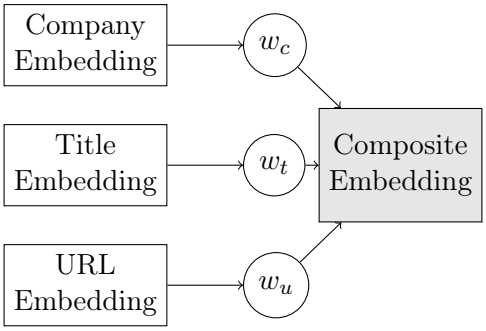

Figure 2: Composite embedding construction showing how company name, position title, and URL embeddings are weighted and summed to form a unified representation.

(2). This matrix is filtered to identify pairs above the similarity threshold, which is configurable to balance between identifying true duplicates and avoiding false positives. Group formation uses a canonical application selection strategy that identifies the most representative application in each group based on similarity score, data completeness, and optionally temporal priority. The grouping algorithm ensures that each application appears in only one group, preventing fragmentation.

The weighted sum uses configurable weights with defaults of $w_c = 0.5$ for company name, $w_t = 0.3$ for position title, and $w_u = 0.2$ for URL, where:

$$w_c + w_t + w_u = 1.0 \tag{3}$$

When attributes are missing, the system handles empty strings by generating zero vectors for those attributes, which effectively reduces their contribution to the composite embedding. The final composite embedding is normalised to unit length regardless of which attributes are present.

4.3 Implementation Details

The system uses sentence transformer models [8], with a default of `all-MiniLM-L6-v2`, to generate embeddings for application attributes. These models are specifically designed for semantic similarity tasks and provide a good balance between accuracy and computational efficiency. Embeddings are generated in batches with a configurable batch size defaulting to 32 items, optimising memory usage and processing time. The similarity threshold is configurable on a 0-100 scale, which is internally converted to a 0-1 scale for cosine similarity computation. Higher thresholds (85-95) reduce false positives but potentially miss legitimate duplicates, and lower thresholds (60-75) capture more duplicates but potentially include false positives.

Performance optimisations include batch processing for embedding generation and similarity computation, caching strategies for embeddings and similarity matrices, and algorithmic optimisations such as early termination for low-similarity pairs, indexing for fast lookups, and approximate nearest neighbour search techniques [9] for very large datasets.

4.4 User Interaction and Benefits

Groups are presented in a hierarchical, collapsible interface that allows administrators to efficiently review large numbers of potential duplicates. The interface provides checkboxes for individual or bulk selection, enabling efficient processing of multiple groups. Administrators can create consolidated job postings, link applications to existing postings, or mark groups as reviewed without taking action. Filtering mechanisms include user ID filtering, linked status filtering, and text-based search functionality.

Application grouping provides several key benefits: reduced data redundancy through identification and consolidation of duplicate entries across all users, enabling the creation of consolidated job postings essential for the jobs board feature, enhanced analytics capabilities from cleaner data that provides more accurate counts and statistics, and improved administrative workflow efficiency by reducing manual work required to prepare data for the jobs board.

5 Company Entity Assignment

5.1 Purpose and Rationale

Company entity assignment addresses the problem of inconsistent company representation that arises from the natural variation in how company names appear across different sources and entry methods. The same organisation may appear under various names due to legal entity variations, brand name differences, user entry inconsistencies, and international naming conventions. The entity system provides a canonical representation for each distinct organisation while maintaining links to all name variations.

For example, the technology company Google might appear in the system as "Google", "Google LLC", "Google Inc.", "Alphabet Inc.", or "Alphabet" depending on where users discovered the job posting and how they entered the information. Without entity normalisation, searches and analytics would be fragmented across multiple name variations, making it impossible to retrieve all applications for a company or generate accurate statistics.

5.2 Entity Model

Company entities consist of a canonical name, website URL, sector classification, and metadata. Companies maintain a many-to-one relationship with entities, providing a single canonical representation while preserving all original name variations. The relationship is reversible, allowing corrections when entity assignments are found to be inappropriate.

5.3 Matching Algorithms

The system identifies duplicate companies using two complementary methods: fuzzy matching and embedding-based matching. The grouping strategy prefers non-custom companies as canonical representations, selects the most common or standardised name, and aggregates metadata to ensure the canonical entity has the most complete information possible.

5.3.1 Fuzzy Matching

Fuzzy matching uses string similarity algorithms to identify similar company names even when they are not identical. The implementation includes comprehensive normalisation that prepares company names for comparison by converting to lowercase, removing punctuation and special characters, standardising whitespace, and handling common abbreviations. The system employs token-based similarity metrics, primarily using token sort ratio which compares strings based on their token sets regardless of order, making it effective for company names where word order may vary. The implementation also supports ratio-based matching (based on Levenshtein distance) and partial ratio matching for substring comparisons. The similarity threshold is configurable on a 0-100 scale, allowing administrators to control matching sensitivity.

5.3.2 Embedding-Based Matching

The system uses sentence transformer models [8] for generating embeddings that capture semantic meaning of company names. The default model is `all-MiniLM-L6-v2`, which provides a good balance between accuracy and computational efficiency. Alternative models such as

`all-mpnet-base-v2` [10] offer higher accuracy at increased computational cost. Embedding generation involves text preprocessing, model encoding to convert text into fixed-size vectors, and L2 normalisation for cosine similarity computation. Since embeddings are normalised to unit length, cosine similarity is computed using the same formula as shown in Equation (2), yielding values in the range $[0, 1]$, with 1 indicating identical semantics and values closer to 1 indicating greater similarity. Caching is employed to improve performance by storing embeddings on disk with cache keys that include the model name and input text.

5.4 Assignment Workflow and Benefits

The assignment workflow begins with detection where the matching algorithm identifies duplicate companies and forms groups. Administrators can select companies individually, select entire groups, or perform bulk selection across multiple groups. Companies can be assigned to existing entities or new entities can be created. Before applying changes, administrators can preview all pending assignments through a preview modal. The commit process applies all changes atomically within a database transaction.

Company entity assignment provides several key benefits: data normalisation that ensures consistent company representation essential for the jobs board, improved search and filtering capabilities through entity-based queries that retrieve all applications and job postings related to a company regardless of name variations, enhanced analytics capabilities from accurate company-level statistics, improved system maintainability through centralised company information management, and scalability through efficient handling of large numbers of company variations.

6 Implementation Architecture

6.1 System Components

The backend is implemented in Python using FastAPI [11], providing RESTful endpoints for all operations. The database layer uses PostgreSQL with Supabase for data persistence. Matching services are implemented as modular components that can be easily extended or replaced. The frontend is implemented in React [12] with TypeScript [13], using the React Context API for global state management and optimistic UI updates with server synchronisation. The database schema includes key tables for applications, companies, company entities, job postings, and the many-to-many relationship table that links applications to job postings.

6.2 Data Flow and Error Handling

The application grouping flow begins when an administrator triggers a grouping operation. The backend loads applications, generates embeddings, computes pairwise similarities, and forms groups based on the similarity threshold. These groups are returned to the frontend for administrator review. Administrators can then select applications and perform actions such as creating job postings or linking to existing postings. When actions are confirmed, changes are persisted to the database.

The entity assignment flow begins when an administrator triggers duplicate company detection. The backend executes the matching algorithm to identify duplicate groups. Administrators select companies and assign them to entities, with pending assignments tracked in the frontend state. When administrators commit changes, the backend creates entities and links companies atomically within a database transaction.

Error handling includes input validation at API boundaries, type checking using TypeScript [13] and Pydantic [14], constraint checking in the database, transaction management for atomic operations, and comprehensive administrator feedback through action logs, error messages, and success confirmations.

7 Evaluation and Results

The framework has been deployed as a production system and is actively used by the LucidTrack development team to maintain data quality and prepare aggregated datasets for platform features. Empirical validation through real-world implementation demonstrates significant improvements in data quality metrics.

The application grouping system successfully identifies and consolidates duplicate entries across all users, enabling the creation of consolidated job postings essential for the jobs board feature. In practice, the system processes thousands of applications, identifying groups of duplicates that would be impossible to detect manually. The composite embedding approach proves effective at matching applications with varying titles, company names, and URLs, capturing semantic similarity that exact or simple fuzzy matching would miss.

Company entity assignment has normalised hundreds of company name variations, creating canonical entities that enable reliable search, filtering, and analytics. The system handles legal entity variations, brand name differences, and user entry inconsistencies, significantly improving the consistency of company representation throughout the database.

The interactive workflow design balances automation with human oversight, allowing administrators to efficiently review and approve algorithmic suggestions. This approach ensures high-quality results while maintaining the flexibility to handle edge cases and domain-specific requirements that automated systems might miss.

Performance characteristics demonstrate the system's scalability: batch processing and caching strategies enable efficient handling of large datasets, with embedding generation and similarity computation completing in reasonable timeframes even for thousands of applications. The modular architecture allows for easy extension and replacement of matching components as new techniques become available.

8 Limitations and Future Work

8.1 Current Limitations

Algorithmic limitations include the fact that embedding models may not capture all semantic nuances, particularly for domain-specific terminology. Threshold selection requires administrator expertise to balance between precision and recall, and performance may degrade with very large datasets. User interface limitations include the requirement for manual review of all matches by administrators, which can be time-consuming for large datasets. There is no automatic confidence scoring to guide administrators in their review decisions, and limited support for edge cases such as company mergers or acquisitions.

8.2 Future Enhancements

Future enhancements could include active learning techniques that improve matching accuracy over time, confidence scores for automated decisions, multi-lingual support for international companies, machine learning-based threshold recommendations, automated grouping suggestions, enhanced visualisation of relationships, real-time duplicate detection, integration with external company databases, and advanced analytics and reporting capabilities.

9 Conclusion

This paper has presented a comprehensive framework for record linkage and entity resolution in job application data, addressing critical data quality challenges that arise when hundreds of users independently discover and save the same job opportunities. The system transforms fragmented, duplicate-ridden application data into clean, structured datasets suitable for jobs

board aggregation through two complementary mechanisms: application grouping and company entity assignment.

The key contributions of this work include: (1) a composite embedding approach that enables multi-attribute similarity matching for accurate application grouping, combining semantic representations of company name, position title, and URL using weighted sums; (2) an entity normalisation system that maintains data integrity while enabling consistent company representation through canonical entities; (3) flexible matching strategies supporting both fuzzy string matching and semantic embedding-based matching with administrator-configurable thresholds; and (4) an interactive administrative workflow design that combines automated detection with human oversight, ensuring high-quality results while maintaining flexibility for edge cases.

Empirical validation through real-world deployment demonstrates significant improvements in data quality metrics, including reduced data duplication, improved consistency, and successful creation of consolidated job postings from user-generated application data. The framework has been deployed as a production system and is actively used by the development team to maintain data quality and prepare aggregated datasets for platform features.

The system's modular architecture and performance optimisations enable scalable processing of large datasets, while the interactive workflow design balances automation with human judgment. Future work will focus on algorithmic improvements that enhance matching accuracy, enhanced automation that reduces the manual review burden, and expanded feature sets that provide additional value for data quality management and jobs board generation. As the platform continues to grow, the data cleansing framework will evolve to meet new challenges and take advantage of advances in matching algorithms and user interface design.

A Technical Specifications

A.1 Configuration Parameters

Configuration parameters include the similarity threshold on a 0-100 scale with a default of 75, the embedding model specified as a sentence transformer model name, the matching method selected as either "fuzzy" or "embeddings", composite weights for company, title, and URL with defaults of 0.5, 0.3, and 0.2 respectively, and batch size for embedding generation with a default of 32 items.

A.2 Database Schema

Key relationships in the database schema include `applications.company_id` linking to `companies.id`, `companies.entity_id` linking to `company_entities.id`, `job_postings.company_entity_id` linking to `company_entities.id`, and `application_job_posting_links` providing a many-to-many relationship between applications and job postings.

References

[1] I. P. Fellegi and A. B. Sunter, "A theory for record linkage," *Journal of the American Statistical Association*, vol. 64, no. 328, pp. 1183–1210, 1969.

[2] P. Christen, *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, 2012.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[4] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[5] M. A. Jaro, "Advances in record linkage methodology as applied to the 1985 census of tampa, florida," *Journal of the American Statistical Association*, vol. 84, no. 406, pp. 414–420, 1989.

[6] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi–sunter model of record linkage," *Proceedings of the Section on Survey Research Methods*, pp. 354–359, 1990.

[7] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[8] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019, pp. 3982–3992.

[9] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[10] K. Song et al., "Mpnet: Masked and permuted pre-training for language understanding," *arXiv preprint arXiv:2004.09297*, 2020.

[11] S. Ramírez, *Fastapi: High performance, easy to learn, fast to code, ready for production*, `https://fastapi.tiangolo.com`, 2020.

[12] I. Meta Platforms, *React – a javascript library for building user interfaces*, `https://reactjs.org`, 2023.

[13] Microsoft, *Typescript: Javascript with syntax for types*, `https://www.typescriptlang.org`, 2023.

[14] S. Colvin, *Pydantic – data validation using python type annotations*, `https://pydantic-docs.helpmanual.io`, 2020.